Introduction to UNIX and the Shell

Hands-On UNIX System Administration DeCal

Lab 1 — 23 January 2012

Introduction

Today we went over the shell, a few common UNIX commands, text streams, substitution, and `vi`. In this lab, you'll get some practice with these tools while learning more about the shell.

This lab is due in hard copy at the beginning of next week's class (that is, by 6:10 PM on the 30th). You can type up your answers in a text file and use `lpr` to print it (your cs198 account comes with 150 free print pages); if you want to use a word processor, try `soffice`. Please **make sure your submission includes your name and your cs198 account name**. If you get stuck, ask for help!

1 The basics

In a sentence or two, precisely describe what each of the following commands does. (You may need to look them up in the `man`ual; some commands are shell builtins that don't have manual entries, but you can always try them out!)

1. `cat`. Fun fact: You'll often see constructions such as `cat README | grep garply`. This is often unnecessary, as many commands will accept files as arguments (`grep garply README`), and using commands this way will win you a Useless Use of Cat award.
2. `ls -a` and `ls -d`.
3. `mkdir -p foo/bar/baz`.
4. `rmdir *`. Hint: `rmdir`'s behavior is not immediately obvious. Be careful.
5. `touch file`.
6. `find ~ -name '.*' 2>/dev/null`.

2 Exit status

All programs in UNIX environments report failure or success when they terminate using a number between 0 and 255, called an *exit status*.[1] (In C, this is determined by the `main()` function's return value, an 8-bit `unsigned int`.)

1. Use the Internet to find a way to display the return value of the last command you ran. Using the Web to find answers is instrumental even for seasoned sysadmins — it's impossible to know *everything* off the top of your head, and web articles often are more easily attacked than dense manpages. In your answer, include your search query and the website where you found your answer.

2. Determine what exit codes are commonly used for success and failure. You might try `wget`'ing the Berkeley website and then a nonexistent URL and comparing exit codes.

3 Job control

The shell wouldn't be very useful if you could only run one program at a time — job control lets you, say, read manpages while in the middle of editing text files, and lets you wrest control of your terminal back from frozen programs. Get started by using `wget` to download a large file from somewhere. (A few megabytes is enough: you just need to buy yourself some time. Your account's disk quota is 512MB, so you may want to clean up with `rm` when you're done.)

1. While your file is downloading, press Ctrl-Z. (This is usually abbreviated to ^Z.) Clearly describe what happens.

2. ^Z will return you to a prompt. Once you're there, figure out what `jobs`, `fg`, and `bg` do. You shouldn't need a search engine or manual — just experiment!

3. Run a few jobs simultaneously and figure out how to switch between them.

4. Start your download again, but this time interrupt it with ^C. Compare the behavior of ^C and ^Z.

5. Programs usually start in the foreground, instead of detaching from your terminal and running in the background. Find a way to start a program in the background.

On a side note, you might have noticed that neither ^C nor ^Z affects your shell. ^D is a quick way to log out of an SSH session or close your terminal. (It sends the EOF, or end-of-file, character, and works as advertised with programs' standard input.)

[1] or *return value*, or *exit code*, or...

4 Editing text

UNIX text editors are not the most user-friendly tools around — some, like `vi`, don't even let you enter text without reading a manual first! Vim (Vi IMproved) has a tutorial, which you can access by running `vimtutor`; you don't need to work through the whole thing, but at least get comfortable with the basics. ^[is equivalent to the Escape key, which will save you a bit of effort.

The shell doesn't (by default[2]) use modal editing, but supports some useful keyboard shortcuts that are worth committing to memory:

- **^U** — delete all text from your cursor to the beginning of the line
- **^K** — delete all text from your cursor to the end of the line
- **^W** — delete the last word (delimited by spaces)
- **^A** — jump to the start of the line
- **^E** — jump to the end of the line

Another useful keyboard shortcut, while not strictly related to editing text, is **^L**, which clears your display.

[2]Many programs use GNU Readline to handle text input; you can configure readline to use Emacs or vi keybindings, depending on which you prefer.